Spring 6-10-2017

WHEN IS AGILE APPROPRIATE FOR ENTERPRISE SOFTWARE DEVELOPMENT?

Gary Spurrier
Bentley University, gspurrier@bentley.edu

Heikki Topi
Bentley University, htopi@bentley.edu

WHEN IS AGILE APPROPRIATE FOR ENTERPRISE SOFTWARE DEVELOPMENT?

Research in Progress

Gary Spurrier, Bentley University, Waltham, Massachusetts, USA, gspurrier@bentley.edu

Heikki Topi, Bentley University, Waltham, Massachusetts, USA, htopi@bentley.edu

Abstract

Using agile methods for enterprise software development (ESD) remains contentious. Advocates of agile and plan-driven methods (i.e., waterfall) argue their respective cases with near evangelical zeal, and recent evidence indicates that waterfall (or some variant) remains a widely used approach. This controversy persists despite strong arguments by Boehm and Turner (2004) recommending a balanced software development approach combining aspects of agile and plan-driven methods, aligned to pro-jects based on each project's fit with agile vs. plan-driven "home ground" characteristics. In this re-search, we hypothesize that Boehm and Turner were fundamentally correct and that neither of the "pure" models will lead to the highest level of project success in all circumstances. This paper de-scribes a research project to study the impact of alignment with a flexible but simple agile vs. hybrid vs. plan-driven approach on ESD outcomes. The discussion includes: 1) Articulating the identifying characteristics of ESD, 2) distilling the essence of plan-driven vs. agile methods along two key dimen-sions, 3) explicating a hybrid method of software development using those dimensions, and 4) extend-ing Boehm and Turner's "home grounds" model to better determine the optimal ESD approach. The discussion includes our planned research questions, data collection and analysis, and hypotheses.

Keywords: Agile Development, Plan-Driven Development, Enterprise Software Development

1 Introduction

Agile software development methods can often deliver project success superior to traditional, plan-driven methods (i.e., waterfall), especially in small projects (The Standish Group, 2015; Hastie and Wojewoda, 2015). What remains contentious is the extent to which agile methods can be successfully used, to use terminology from Boehm and Turner (2003, 2004), in contexts outside of the agile meth-ods' "home ground." The context we are particularly interested in this research is enterprise software development (ESD). The home grounds model states that there are different characteristics corre-sponding to agile and to plan-driven methods under which each type of method is most likely to suc-ceed (Boehm and Turner, 2004, p. 51). In Table 1, we present a refined and extended version of Boehm and Turner's original home grounds model based upon ideas from subsequent literature (Turk et al., 2005; Leffingwell, 2007; Ågerfalk et al., 2009; Sheffield and Lemétayer, 2013). In this table, the Industry and Organization characteristics in which a software development project is embedded are in the first table section. The second section includes the Project and Application software characteristics that define a given software release or initiative. (Bolded text is explained in the next section.)

Characteristic	Agile Home Ground	Plan-Driven Home Ground
Overarching context: Industry/Organization Characteristics		
Goals & Values	• Rapid, responsive delivery of value	• Predictable, high assurance delivery
Industry	• Turbulent/rapidly evolving	• Stable/mature
Organization	• Agile organization valuing freedom/empowerment/chaos	• Plan-driven organization valuing policies/procedures/control
IT Team contends with: Project/Application Characteristics		
Customers/ Product Owners	• Customers/Product Owners: Few, dedicated, co-located	• **Customers/Product Owners: Many, not dedicated, not co-located**
Software Requirements	• Small/flexible scope • Low development interdependence • Low clarity • Low stability over time • Single project focused	• **Large/fixed scope** • High development interdependence • High clarity • High stability over time • **Project portfolio/organization-focused**
Software Application	• "Greenfield" or small code base • Non-strategic/non-mission-critical • Low security/safety risk • No need for intentional architecture	• **Large code base and/or "legacy" app** • **Strategic/mission-critical** • **High security/safety risk** • **High need for intentional architecture**
IT Team	• IT Team: Small, generalists, co-located, high-performing, stable/cohesive, using tacit/shared informal knowledge	• **IT Team(s): Large, specialists, multiple locations and time zones, few assumptions regarding performance levels, unstable/new, needing documentation for knowledge transfer**

Table 1. Extended Home Grounds Model: Industry/organizational/software characteristics.

2 Background

2.1 ESD in context of home grounds

It is straightforward to establish that software development in an enterprise context falls outside of the agile home ground. To begin, given the intrinsically risky nature of software development, organizations tend to avoid risk by using commercial off the shelf (COTS) software for functionality that is relatively standardized and thus not contributing to strategic differentiation (Slaughter et al., 2006).

Thus, organizations tend to focus their own software development efforts on systems supporting functionality unique to their industry vertical and providing strategic differentiation. For example, an Internet retailer would likely focus on its web site, fulfilment, and customer profiling, while an insurance broker would focus on policy renewals and risk analytics. Given this, we characterize enterprise software developed within organizations as generally having the following distinguishing characteristics:

- **Strategic & mission critical:** Software key to strategic differentiation and ability to compete.
- **Complex, organization-specific functionality:** Typically not attainable via COTS software.

Because of these two factors, enterprise software also corresponds to some or all of the following:

- **Serving many users:** Across the organization, often in multiple roles and units.
- **High risk:** From handling customer data, facing the Internet, and/or supporting human safety.
- **Needing planned, intentional architecture:** To support industrial-strength reliability, scalability, extensibility, criticality, and large amounts of complex, persistent data.
- **High budget, large IT staff, and high visibility:** Arising from all of the factors above.

By mapping these factors to Table 1 (shown in bold font), it is evident—perhaps not surprisingly—that enterprise software development tends to fall solidly in the plan-driven home ground column.

Given this, one might wonder why a number of authors of recent, popular trade books on this subject (e.g., Larman and Vodde, 2009, 2010; Leffingwell, 2007, 2011; Gruver, Young, and Fulghum, 2013; Gruver and Mouser, 2015) have forcefully advocated for using agile approaches in most large-scale ESD projects. Yet, despite this advocacy, plan-driven approaches such as waterfall or variants such as iterative development within a waterfall process ("Water-Scrum-Fall") remain prevalent today (Nelson and Morris, 2014; West et al., 2011).

Part of the answer lies in the evangelistic zeal of some agile advocates, who see in agile a way to make software development not just more effective, but also more humane (Boehm and Turner, 2004, p. 4).

Still, seen from a more instrumental perspective, applying agile approaches to ESD seems puzzling. By delaying requirements elaboration until the "last responsible moment" (Leffingwell, 2007, p. 191), agile appears to contradict the principle of reducing uncertainty by as much as possible as early as possible, which has been a core idea in the management literature for decades (Thompson, 1967, p. 159). Conversely, we can discern why plan-driven approaches try to reduce requirements uncertainty as early in the software development process as possible, especially with large scope. To make sense of this, we need first to distil the essence of agile vs. plan-driven methods across two key dimensions, and then analyse their associated, underlying cost and value assumptions.

2.2 The essence of agile vs. plan-driven methods

Specific agile and plan-driven methods differ in their details. For example, in agile, XP specifies the use of pair programming, which is not an element of Scrum or most other methods such as DSDM or FDD (Leffingwell, 2007, p. 38). Similarly, in plan-driven, the waterfall differs from earlier stage models in the incorporation of feedback loops between successive stages and a greater emphasis on prototyping during requirements (Boehm, 1988), while the V-Model emphasizes earlier development of test cases (Balaji and Murugaiyan, 2012).

However, at their core, agile and plan-driven approaches can each be fundamentally characterized in meaningful, straightforward ways. For example, virtually all agile approaches appear to embody the principle of iterative development and concurrent testing, with detailed elaboration of requirements scope delayed until the iteration in which the requirements will be built (Leffingwell, 2007, pp. 81-82). Conversely, all plan-driven approaches emphasize linear development with requirements elabora-tion completed prior to that development, commonly and interchangeably termed "Big Requirements Up Front (BRUF)" (Ambler, 2014), "Big Design Up Front (BDUF)" (Boehm and Turner, 2004, p. 55), or "Big Up Front Design (BUFD)" (Leffingwell, 2007, p. 30). See Table 2.

Dimension	Plan-Driven Approach	Agile Approach
Requirements	**BRUF/BDUF/BUFD:** • Fixed scope up-front • Fully elaborated	**Emergent requirements:** • High-level "stories" reprioritized each sprint • Elaborated during iterative development
Development	**Non-iterative:** • Single long-sequence • Testing on delivery	**Iterative:** • Concurrent development & test • Demonstrations identify revisions needed

Table 2. Essential dimensions of software development: plan-driven vs. agile.

This 2X2 matrix implies the possibility of two other, hybrid software development approaches:

- **BRUF with iterative development:** This is the mixed, balanced approach we explicate below.
- **Emergent requirements with non-iterative development:** Upon reflection, this can be seen as an illogical, "degenerate case" that does not need to be considered further.

Thus, Table 3 defines three essential software development approaches: plan-driven, hybrid, and agile.

Software Phase	Plan-Driven		Hybrid		Agile	
Current State Analysis • Make explicit • Resolve differences	BRUF	**Current State**	BRUF	**Current State**	Planning	**Story Backlog** • Prioritization • Minimum Viable Product
Requirements Analysis ("What") • Business goals • New features		**Future State** • Fixed scope		**Future State** • Scope (maximum boundaries vs. minimum features)		
Functional Requirements ("How -- User View") • Business Process Design (BPD) • Data model • Logic • UI/UX		**Functional Specification** • BPD • Use cases • ERD/Domain models • UI/UX Designs		**Functional Specification** • BPD • Use cases • ERD/Domain models • UI/UX Designs • Test cases	Iterative Development	**Iterative Functional Requirements** • Lightweight design methods • Emergent from iterative development
Technical Design/ Development ("How -- Developer View") • Class designs • Other UML or other technical designs	Development	**Technical Design/ Development** • Class designs • Development • Unit tests	Iterative Development	**Technical Design/ Development** • Revised "within boundaries" functional designs • Tech designs per Plan-Driven		**Technical Design /Development** • Lightweight design methods • Emergent from iterative development
Testing	Testing	**System Tests, User Acceptance Tests**		**Concurrent Tests**		**Concurrent Tests**

Table 3. Essential plan-driven vs. hybrid vs. agile software development approaches.

2.3 The hybrid approach and relationship to plan-driven and agile

Table 3 that illustrates that aspects of the plan-driven and agile approaches can be effectively combined into a hybrid approach, combining the BRUF of plan-driven with the iterative construction of agile. This approach is in fact commonly used today (West et al., 2011), given that it combines some of the advantages of significant up-front requirements and design with the ability to make revisions or "course corrections" from frequent customer feedback during software construction. In hybrid, it is all (or most) detailed requirements are created prior to starting the construction phase of the project. However, it is also true that these detailed requirements are subject to revisions as construction iterations are completed, software is demonstrated to customers, and those customers provide feedback. Thus, hybrid projects will in reality complete only a portion of their detailed requirements "up front," with more SA&D requirements work occurring on an on-going basis after construction commences.

2.4 Requirements approach: underlying assumptions

Three key assumptions underlie the BRUF vs. agile requirements approach selection decision:

- **Scope clarity:** Plan-driven assumes BRUF works because a well-defined set of requirements can be created with proper time and effort (Leffingwell, 2007, p. 20). In contrast, agile holds that BRUF fails, because accurately envisioning requirements prior to creating working code is nearly impossible, especially given that deploying software, itself, tends to change requirements.
- **Scope stability:** Numerous agile sources (e.g., Cockburn and Highsmith, 2001; Augustine et al., 2005; Sarker and Sarker, 2009) make broad claims that requirements are changing more rapidly in recent years due to an increasingly dynamic business environment. This explains why agile eschews the uncertainty-reducing value of BRUF, as BRUF specifications would rapidly decline in fidelity to actual business requirements as development continues over time. Thus, agile argues for elaborating requirements at the "last responsible moment." In contrast, plan-driven assumes requirements are reasonably stable, preserving the value of BRUF during development.
- **Cost of late requirements changes/development interdependence:** Plan-driven assumes that change costs accelerate as development progresses, thus underlining the need for BRUF to avoid them (Leffingwell, 2007, p. 32). In contrast, agile assumes the cost of making changes remains nearly level, implying little economic downside in elaborating at the "last responsible moment" and thereby lowering the value of BRUF. We posit that a key determinant of "cost of changes" is the increasing "development interdependence" of project features. This occurs where scope is both large and intricate. Indeed, there are two non-mutually exclusive situations that can generate development interdependence in this way:
    - **Large complex scope within a single application system:** This can lead to needing to split stories into smaller, linked predecessor/successor stories built over a series of sprints (Leffingwell, 2011, pp. 110-114; Wake, 2012). In this case, late changes to an early story may have ripple effects through its successor stories, leading to the plan-driven view of accelerating costs over time. Conversely, if stories are "development independent," then costs of late changes will remain relatively low, as per agile.
    - **Intersystem "release train" interactions:** If the project team must deliver its overall solution by combining capabilities with applications supported by other teams, then those dependencies increase development interdependence. This need to synchronize "release trains" is especially pertinent when the other teams are operating using different approaches, incompatible requirements, cadences, and release dates (Leffingwell, 2007, pp. 237-247).

The differing assumptions and implications of plan-driven and agile approaches are shown in Figure 1.

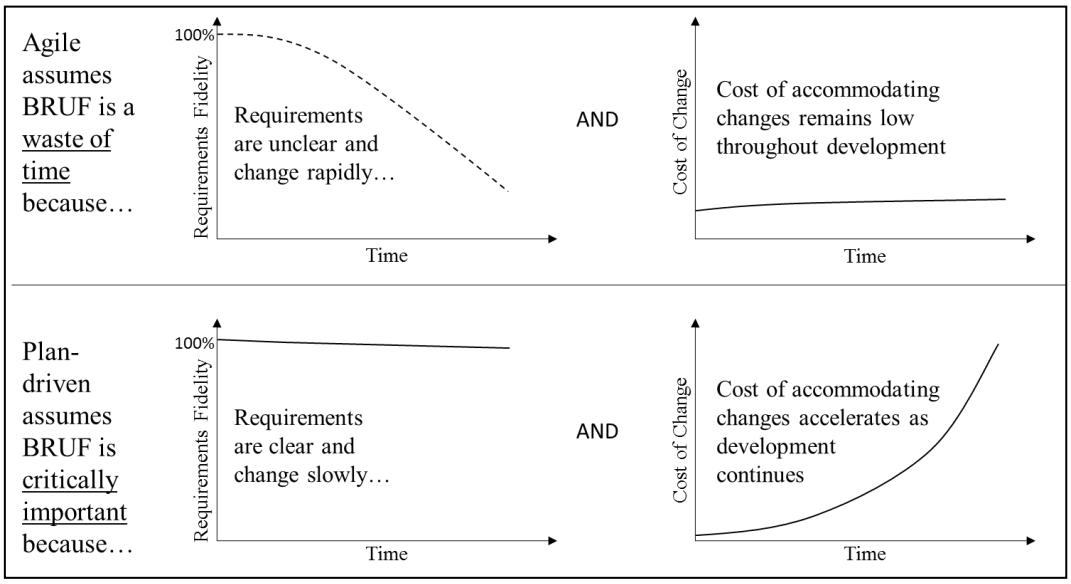

Figure 1. Agile vs. plan-driven assumptions determining the value of BRUF.

Which camp—agile or plan-driven—is correct with respect to these assumptions? We suggest that the answer is "neither." Rather than being simply true or false in all circumstances, each of these "assumptions" are variable characteristics that must be assessed separately for each project.

2.5 Iterative development: an inherently better approach?

The above discussion summarizes arguments why BRUF vs. emergent requirements may be appropriate in different circumstances. In contrast, with respect to development, a review of the literature provides virtually no arguments supporting non-iterative development over iterative development. Rather, the literature appears to argue that iterative is intrinsically superior to non-iterative. This is because iterative enables frequent customer feedback and refinements to requirements after each development iteration (Rubin, 2013, pp. 33-34), while in non-iterative, the customer is only able to view and provide feedback at the end of the development process. Note that iterative advantages appear to apply to both hybrid and agile, even given that in hybrid, BRUF creates overall functionality boundaries that limit changes to the general scope of the project (but not limiting refinements within that general scope) (Turk et al., 2005). This leads to a key question: if iterative development is inherently and generally superior to non-iterative, does it follow that the traditional "pure" plan-driven approach should never be used? If so, our choice of approach to executing ESD is reduced to two options: hybrid vs. agile. This is a question to be addressed in our research (specifically, RQ3 and H1, below).

3 An Extended Theory of Method Selection for ESD

As noted at the outset, our approach is to take the theory of Boehm and Turner (2003) and extend it to accommodate the insights offered above. This original theory recognized the complex, multi-faceted nature of software development projects, depicting five variables driving a project's alignment with either the plan-driven vs. agile home approaches ("low-to-high" indicates that a high value corresponds to plan-driven, while "high-to-low" indicates that a low value corresponds to plan-driven):

- **Size:** number of personnel on the team (low-to-high).
- **Personnel:** measurement of percent of team that are highly skilled and capable (high-to-low).
- **Dynamism:** percent of requirements changing per month (high-to-low).

- **Criticality:** loss due to impact of defects (low-to-high).
- **Culture:** percent of team thriving on chaos/empowerment/freedom vs. order/policies/procedures (low-to-high).

Comparing these five dimensions to those developed in the discussion above, it is clear that "Dynamism" corresponds closely as the inverse of "scope stability" from section 2.3. However, section 2.3's "scope clarity" and "development interdependence" (along with "scope size" and "scope complexity," which drive development interdependence) are not explicitly represented in the original model, and, still, they would seem to have a major impact on the selection of approach. Thus, given their consistency with the home grounds models, we have included these dimensions as rows in the home grounds table (Table 1, items not bolded in the "Project/Application Characteristics" section).

We have grouped these factors into a **"Method Selection Dimensions"** group, while making the following additional refinements to the original model:

- **"People-Based Sources of Difficulty" group:** Taking the original model's "personnel," "size," and "culture" dimensions, we have created this group adding key dimensions that have emerged in recent writings as issues when operating at scale: location, and customer team characteristics.
- **"Need for Intentional Architecture" group:** Taking the original model's "criticality" dimension and creating this group including the non-functional requirement dimensions that often arise in enterprise software: "reliability," "scalability," "extensibility," and "criticality."

With these refinements, Figure 2 presents the revised model. Overall, based on our analysis of agile vs. plan-driven assumptions, above, our method selection model emphasizes scope characteristics over other issues, such as the original theory's "Size," "Personnel," "Criticality," and "Culture."

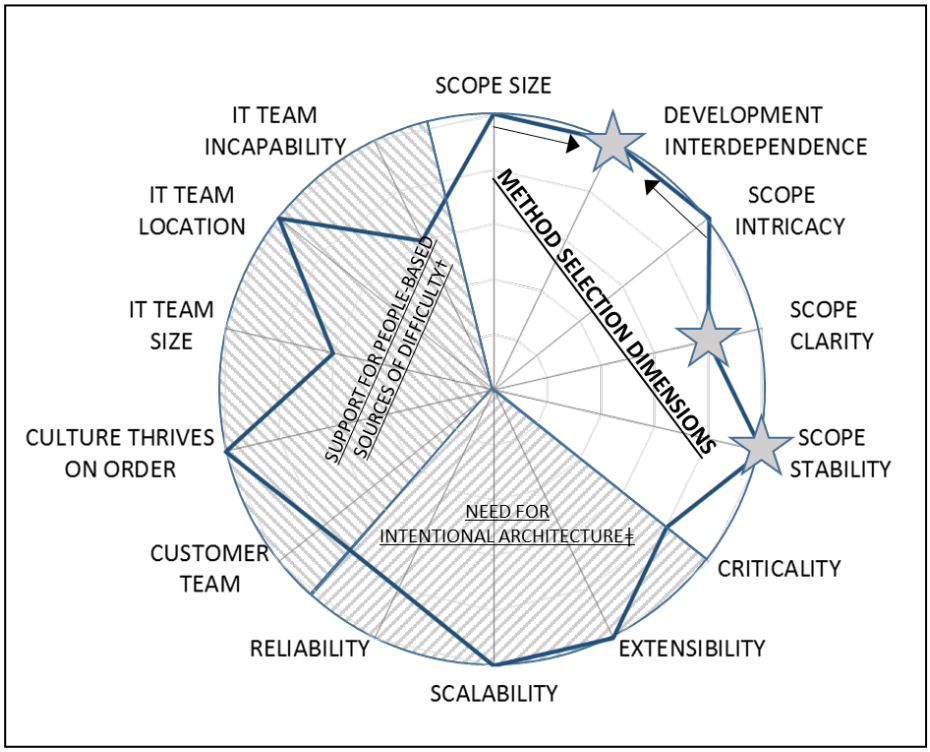

Figure 2. Extended model; stars show key theory dimensions determining optimal method selection. Values near the perimeter suggest hybrid or plan-driven, rather than agile.

4 Research questions, hypotheses, target sample, and limitations

We are focused on answering the following research questions (RQs):

- **RQ1: Is Plan-driven vs. Hybrid vs. Agile an appropriate taxonomy for ESD?** In other words, do practitioners recognize their own project practices as being described by this taxonomy? This would be important both as a precursor to answering the remaining RQs, but also as a way to establish a relatively simple but meaningful taxonomy of key software development approaches.
- **RQ2: Are key ESD project characteristics associated with method selection in a way predicted by the theory?** These include that high values of "scope stability," "scope clarity," and "development interdependence" predict selection of either a Hybrid or Plan-Driven approach emphasizing BRUF, while low values of these would predict select of an Agile approach. Regarding specifically "development interdependence," we also plan to determine if "development interdependence" that construct correlates with high values of "scope size" and "scope intricacy."
- **RQ3: Do ESD projects using the method predicted by the theory experience higher rates of project success?** "Project success" will be the dependent variable, measured using multiple dimensions, including the traditional "iron triangle" of budget, timing, and quality, but also, importantly, business success, specifically "fitness for purpose" (de Wit, 1988; Baccarini, 1999; Nelson, 2005).

Specific hypotheses (HX) to test include the following:

- **H1: Iterative development increases ESD project success:** Based on the factors discussed in Section 2, we expect to find a main effect for this.
- **H2: The key ESD project characteristics predict method selection:** Specifically,
  - **Agile:** Low scope clarity, scope stability, and development interdependence.
  - **Hybrid or Plan-Driven:** High scope clarity, scope stability, and development interdependence.
- **H3: Use of the method compatible with the theory increases ESD project success:** Primarily using the business-oriented "fitness for purpose" (see RQ3, above), as that is seen as the most important ultimate outcome in the relatively short-term, and at times superseding (at least in overall successful projects) minor deficiencies in budget, timing, and quality. However, we will also check for relationships between the method and other measures of project success.

We will test these hypotheses using data from real organizations and projects. We are now validating our constructs and research questionnaire via semi-structured 30 to 60 minute interviews of IT practitioners. The finalized questionnaire will then be posted as a link on a public website for project managers (our interviews suggest that only project managers possess the knowledge to consistently provide complete responses). This approach will limit our findings in three key ways. First, this is correlational, rather than experimental, research, thus limiting inferences of causality. Second, the focus on project managers for responses could skew our findings to views of that particular role, excluding views of senior sponsors, other IT team members, and front-line business users. Third, our findings will need to be interpreted within the sphere of a largely English-speaking, western cultural context.

5 Summary

In conclusion, we intend to conduct this research in a manner that combines scientific rigor with a high degree of relevance to IT practitioners, educators, and students. The outcomes could result in a significant contribution to the field, including fulfilling Boehm and Turner's desire to guide those people perplexed by the "method wars" to a more balanced and effective approach appropriate to each project circumstance, resulting in a significant increase in the rate of project successes over time.

References

Ågerfalk, P. J., B. Fitzgerald and S. A. Slaughter. (2009). "Introduction to the Special Issue—Flexible and Distributed Information Systems Development: State of the Art and Research Challenges." *Information Systems Research*, 20(3), 317–328.

Ambler, S. (2014). "Examining the 'Big Requirements Up Front (BRUF) Approach.'" URL: http://agilemodeling.com/essays/examiningBRUF.htm (visited on November 18, 2016).

Augustine, S., B. Payne, F. Sencindiver and S. Woodcock. (2005). "Agile project management: steering from the edges." *Communications of the ACM*, 48(12), 85–89.

Baccarini, D. (1999). "The Logical Framework Method for Defining Project Success." *Project Management Journal*, 30(4), 25-32.

Balaji, S., and Murugaiyan, M.S. (2012). "Waterfall vs. V-Model vs. Agile: A Comparative Study on SDLC." *International Journal of Information Technology and Business Management*, 2(1), 26-30.

Boehm, B. (1988). "A Spiral Model of Software Development and Enhancement." *IEEE Computer*, 21(5), 61-72.

Boehm, B., and R. Turner. (2003). "Using Risk to Balance Agile and Plan-Driven Methods." *Computer*, 36(6), 57-66.

Boehm, B. W. and R. Turner. (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston: Addison-Wesley Professional.

Cockburn, A. and J. Highsmith. (2001). "Agile software development, the people factor." *Computer*, 34(11), 131–133.

De Wit, A. (1988). "Measurement of project success." *International Journal of Project Management*, 6(3), 164–170.

Gruver, G. and T. Mouser. (2015). *Leading the Transformation: Applying Agile and DevOps Principles at Scale*. Portland, OR: IT Revolution.

Gruver, G., M. Young and P. Fulghum. (2013). *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*. Upper Saddle River, NJ: Pearson Education.

Hastie, S., and S. Wojewoda. (2015). "Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch." URL: https://www.infoq.com/articles/standish-chaos-2015 (visited on October 27, 2016).

Larman, C. and B. Vodde. (2009). *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Boston: Pearson Education.

Larman, C. and B. Vodde. (2010). *Practices for Scaling Lean & Agile development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. Boston: Pearson Education.

Leffingwell, D. (2007). *Scaling Software Agility: Best Practices for Large Enterprises.* Boston: Pearson Education.

Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Boston: Pearson Education.

Nelson, R. R. (2005). "Project retrospectives: Evaluating project success, failure, and everything in between." *MIS Quarterly Executive*, 4(3), 361–372.

Nelson, R. R. and M. G. Morris. (2014). "IT Project Estimation: Contemporary Practices and Management Guidelines." *MIS Quarterly Executive*, 13(1), 15-30.

Rubin, K. S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Upper Saddle River, NJ: Addison-Wesley.

Sarker, S. and S. Sarker. (2009). "Exploring Agility in Distributed Information Systems Development Teams: An Interpretive Study in an Offshoring Context." *Information Systems Research*, 20(3), 440–461.

Sheffield, J. and J. Lemétayer. (2013). "Factors associated with the software development agility of successful projects." *International Journal of Project Management*, 31(3), 459–472.

Slaughter, S., L. Levine, B. Ramesh, J Pries-Heje, and R. Baskerville. (2006). "Aligning Software Processes with Strategy." *MIS Quarterly*, 30(4), 891-918.

The Standish Group. (2015). "Chaos Report 2015." URL: https://www.standishgroup.com/store/services/chaos-report-2015-blue-pm2go-membership.html (visited on June 26, 2016).

Thompson, J. D. (1967). *Organizations in Action: Social Science Bases of Administrative Theory*. New York: McGraw-Hill.

Turk, D., R. France and B. Rumpe. (2005). "Assumptions underlying agile software development processes." *Journal of Database Management*, 16(4), 62–87.

Wake, B. (2012). "Independent Stories in the INVEST Model." URL: http://xp123.com/articles/independent-stories-in-the-invest-model/ (visited on October 27, 2016).

West, D., M. Gilpin, T. Grant and A. Anderson. (2011). "Water-scrum-fall is the reality of agile for most organizations today." *Forrester Research*.